Approximation Algorithms for Charging Station Placement for Mobile Robots

Tanmoy Kundu[1] and Indranil Saha[2]

***Abstract*— Optimal placement of charging stations in a workspace is a crucial problem to address, for efficient operation of battery-driven mobile robots. When the battery charge of a robot reaches a certain threshold, the robot must be able to reach a nearby charging station to recharge its battery. In this paper, we deal with two different versions of the optimization problem related to the optimal placement of charging stations in a robot workspace. The first problem is formulated to find an optimal number of charging stations given a battery threshold deciding the need to move to a charging station, and the second problem finds an optimal battery threshold for a given number of charging stations. Both the problems involve finding the locations of charging stations, such that from any obstacle-free location at least one charging station is reachable with at most threshold amount of battery charge remaining with the robot. In this paper, we prove these optimization problems to be NP-hard, i.e., computationally intractable. To handle intractability of the above minimization problems, we design two polynomial-time approximation algorithms to find near-optimal solutions. Our algorithms achieve significantly high scalability without compromising the quality of the solution beyond a certain factor of the optimal solution. Experimental results show that our algorithms run order-of-magnitude faster than a recently proposed Satisfiability Modulo Theory (SMT)-based approach and maintain solution quality within the theoretical bounds on the optimal solution.**

I. Introduction

Indoor mobile robots are widely used in many applications, including manufacturing and materials handling in factories and warehouses, surveillance, and carrying out domestic chores. Most of these applications require the robots to carry out their tasks perpetually. However, as these robots are generally battery-powered, they need to suspend their work from time to time and move to a battery charging location to recharge their batteries. Several approaches have been proposed in the past to ensure persistent power supply to mobile robots, such as docking based autonomous recharging (e.g., [1], [2], [3], [4], [5], [6]), tethering with a continuous power supply (e.g., [7], [8], [9]), and exploiting natural power resources (e.g., [10], [11], [12]). To optimize performance, some recharge scheduling techniques have been proposed with mobile rechargers in the past (e.g., [13], [14]).

The efficiency of the mobile robots depends on the locations of the charging stations. If the charging stations are available in convenient locations, the off-time of the robots may reduce significantly. On the other hand, if the charging stations are not placed in proper locations in the workspace, it may affect the performance of the system as many robots may go down as they cannot reach a charging station with their critically low battery charge. However, due to the complex structure of the workspace and the dynamical constraints of the robot control, solving the *Charging Station Placement Problem* (CSPP) becomes challenging.

In a robot workspace, the charging stations should be placed in a way that from any obstacle-free location in the workspace, at least one charging station is reachable within a specified distance. When the energy available to the robot goes below a pre-decided threshold, the robot needs to abort its mission and reach a nearby charging station to recharge its battery. Assuming execution of each motion primitive consumes a unit amount of battery charge, we use the terms "number of robot transitions" and "amount of battery charge" interchangeably. In this paper, two optimization versions of the CSPP have been studied. The first one is FINDNCS, which finds the optimal number of charging stations in a workspace, ensuring that a robot can reach at least one of the charging stations within a given number of transitions (battery charge threshold). The second version is FINDD, which finds the optimal battery charge threshold for a given number of charging stations in a workspace. The charging station placement may be restricted to a subset of obstacle-free locations, i.e., to a set of potential charging station locations. Both versions handle optimal placement of charging stations in the workspace, such that a robot can reach a charging station from any location in the workspace with a threshold amount of battery charge remaining with it.

In this paper, we establish the *computational intractability* of FINDNCS and FINDD by proving those problems to be NP-hard. We convert these problems into decision problems and apply polynomial-time reductions from well-known NP-hard problems, in order to establish their NP-hardness. We design two polynomial time approximation algorithms to solve the NP-hard problems. We derive approximation ratios of the algorithms in order to specify the closeness of the generated solutions to the optimal solutions. Our design of algorithms incorporates ideas from the approximation algorithms for finding the *Set Cover* and *Dominating Set* in order to solve the versions of the CSPP. Our approximation algorithms for FINDNCS and FINDD provide approximation ratios of $\ln(n+1)$ (n is the number of obstacle-free locations in the workspace) and 2 respectively.

We carry out experiments with five different workspaces and three different robot dynamics and compare the performance of our approximation algorithms with different variants of SMT-based algorithms proposed in [15]. In terms of computation time, our algorithms run order-of-magnitude

[1]Tanmoy Kundu is with the Department of Aerospace Engineering, Technion - Israel Institute of Technology, Israel. This work was carried out when he was with the Department of Computer Science and Engineering, Indian Insitute of Technology Kanpur, India. tanmoy1040@campus.technion.ac.il.

[2]Indranil Saha is with the Department of Computer Science and Engineering, Indian Insitute of Technology Kanpur, India. isaha@cse.iitk.ac.in .

faster than the SMT-based techniques proposed in [15]. Also, the quality of the solutions generated by our algorithms is at par with the SMT algorithms.

In summary, we make the following contributions:

- We formally prove NP-hardness of the optimization problems FINDNCS and FINDD (Sections III and IV).
- To solve the above NP-hard problems, we present two polynomial-time approximation algorithms that take into account complex robot motion primitives and workspace designs. We provide guarantee on the quality of the solutions obtained by our algorithms (Section V).
- We evaluate our algorithms exhaustively with several workspace benchmarks and robot dynamics. The experimental results show the closeness of the generated solutions to optimal solutions and the high scalability of our algorithms (Section VI).

II. PROBLEM

A. Preliminaries

1) Workspace: We assume that the robots operate in a 2-D workspace which is represented as a 2-D occupancy grid map. The grid decomposes the workspace into square-shaped blocks, which are assigned unique identifiers to represent their locations in the workspace. We denote the set of locations in the workspace by W and the set of locations covered by obstacles by O. The set of obstacle-free locations in the workspace is denoted by F, where $F = W \setminus O$.

2) Robot State: The state of a robot σ consists of (1) its position in the space, $\sigma.pos$ (which determines a unique block in the occupancy grid) and (2) its velocity configuration, $\sigma.vel$, which represents the current magnitude and direction of the velocity of the robot. We denote the set of all velocity configurations by U, and assume that it contains a value v_0 denoting that the robot is stationary.

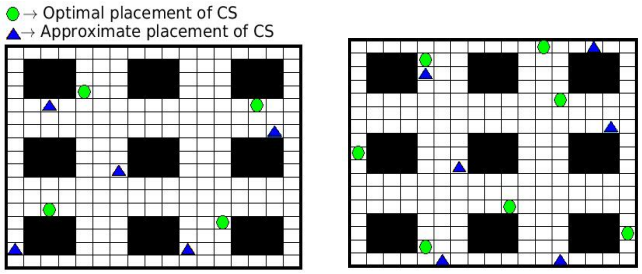
3) Motion Primitives: We capture the motion of a robot using a set of motion primitives Γ. We assume that the robot moves in an occupancy grid in discrete steps of τ time units. A motion primitive is a short controllable action that the robot can perform at any time step. A robot can move from its current location to a destination location by executing a sequence of motion primitives. In this paper, we use the word "step" to denote a transition effected by a motion primitive.

We assume that the robots are battery-powered. The amount of battery charge available to the robot is denoted by the number of motion primitives the robot will be able to execute with the available battery charge.

4) Graph representation: In this work, we define the versions of CSPP in the graph paradigm. We construct a directed graph $G = (V, E)$ that captures all possible state transitions of the robot in the workspace. The set of vertices V in G corresponds to the set of all possible $\langle position, velocity \rangle$ pairs, or the states S of the robot. Mathematically, $S = F \times U$. We define a mapping $f : S \rightarrow V$ that maps each state of the robot to a unique vertex in the graph G. The edges $E \subseteq V \times V$ in G capture all possible transitions between two vertices (i.e., states in S). For any two vertices $u, v \in V$, there is a directed edge $(u, v) \in E$ if and only if the following holds:

1) There exists some states $\sigma_1 \in S$ and $\sigma_2 \in S$ such that $f(\sigma_1) = u$ and $f(\sigma_2) = v$, and
2) There exists a motion primitive $\gamma \in \Gamma$ that drives the transition from σ_1 to σ_2, i.e., $\sigma_1 \xrightarrow{\gamma} \sigma_2$.

The graph thus constructed contains edges with all possible transitions induced by any motion primitive of a given type of robot. We assign each edge a weight of 1 because applying one motion primitive counts to one transition. Also, we denote the set of *potential vertices* to place the charging stations as $\widehat{CS}$. Generally, $\widehat{CS}$ is pre-decided and a subset of the obstacle-free vertices in the workspace, i.e., $\widehat{CS} \subseteq V$.



(a) FINDNCS: Charging station locations for fixed $d = 6$

(b) FINDD: Charging station locations for fixed $ncs = 7$

Fig. 1. Charging station (CS) locations are shown for: (a) fixed $d = 6$ and (b) fixed $ncs = 7$, for Warehouse 17×17 workspace and Turtlebot motion primitives. Charging station placements for optimal solution (circles) and approximate solution (triangles) are shown for both (a) and (b).

B. Problem definition

With the necessary preliminaries, the two versions of the CSPP are defined below.

*Problem 2.1 (*FINDNCS*):* Given a directed graph $G = (V, E)$ with a set of potential charging stations $\widehat{CS} \subseteq V$ and a battery threshold (number of transitions) d, Minimize the number of charging stations ncs and find the charging station vertices $CS = \{v_{c_1}, \ldots, v_{c_{ncs}}\}$ from $\widehat{CS}$ such that at least one vertex in CS is reachable from any vertex $v \in V$ by traversing a path of length at most d.

*Problem 2.2 (*FINDD*):* Given a directed graph $G = (V, E)$ with a set of potential charging stations $\widehat{CS} \subseteq V$ and number of charging stations ncs, Minimize the value of the battery threshold d and find the charging station vertices $CS = \{v_{c_1}, \ldots, v_{c_{ncs}}\}$ from $\widehat{CS}$ such that at least one vertex in CS is reachable from any vertex $v \in V$ by traversing a path of length at most d.

In the next sections, after illustrating an example of the CSPP, we prove that the two above-mentioned problems are NP-Hard. Subsequently, we provide two polynomial-time approximation algorithms to solve those problems.

C. Motivating Example

Consider a Turtlebot operating in a 17×17 workspace (Figure 1) where optimal and approximate solutions for CSPP are shown. Optimal solutions are obtained by a state-of-the-art algorithm in [15], whereas the approximate solutions are obtained by our algorithms proposed in this paper.

A Turtlebot can move one position straight or diagonally in any direction. In Figure 1(a), for the battery threshold (or, the number of transitions) $d = 6$, a charging station must be reachable within at most 6 transitions from any obstacle-free

location in the workspace. The optimal number of charging stations (ncs) is 4, which is obtained by executing the state-of-the-art algorithm in [15] in $47\,\text{min}$. Our algorithm finds the approximate solution as $ncs = 5$ in $43\,\text{s}$.

On the other hand, for fixed $ncs = 7$, optimal value of d is 5, whereas our algorithm gives $d = 6$ (Figure 1(b)). Obtaining a guaranteed optimal solution for this instance takes $106\,\text{min}$, whereas our algorithm obtains the approximate solution in $5\,\text{s}$. In this case, we are allowed to have at most 7 charging stations, but our algorithm finds the d value with 6 charging stations. In Section V, we prove that the solutions found by our algorithms lie within certain bounds on the optimal solutions.

III. Finding Minimum Number of Charging Stations is NP-hard

In this section, we prove Problem 2.1 to be NP-hard. The decision version of FindNCS is as follows:

*Problem 3.1 (*FindNCS_dec*):* Consider a directed graph $G = (V, E)$, a set $\widehat{CS} \subseteq V$ of potential charging station locations, and a positive integer d which represents the battery threshold in terms of the number of transitions by a robot in G. Given a positive integer k, will it suffice to install at most k number of charging stations $CS \subseteq \widehat{CS}$ such that at least one vertex in CS can be reached from any vertex $v \in V$ by traversing a path of length at most d? We denote an instance of the problem by $\langle G, \widehat{CS}, d, k\rangle$.

We choose the *Dominating Set Problem* (Domset), an NP-hard optimization problem, to prove the NP-hardness of FindNCS_dec.

*Problem 3.2 (*Domset *[16]):* Consider a graph $G = (V, E)$. A dominating set in G is a subset $S \subseteq V$ such that every vertex $v \in V$ is either in S or adjacent to a vertex in S. Let $\texttt{domset}(G)$ denote a dominating set in G. Find the minimum cardinality $\texttt{domset}(G)$ in G.

The decision version of Domset is as follows:

*Problem 3.3 (*Domset_dec*):* Given a graph $G = (V, E)$ and a positive integer k, does there exist a dominating set $\texttt{domset}(G)$ in G, such that $|\texttt{domset}(G)| \leq k$? We denote an instance of this decision problem as $\langle G, k\rangle$.

Theorem 3.4: **FindNCS_dec is NP-hard.**

Proof: We show a polynomial time reduction from the NP-hard problem Domset_dec to FindNCS_dec. Given a problem instance $I : \langle G, k\rangle$ of Domset_dec, we create an instance $I' : \langle G', \widehat{CS}, d, k\rangle$ of FindNCS_dec as follows. G in I is an undirected graph, whereas FindNCS_dec accepts directed graphs. We transform the undirected graph $G = (V, E)$ into a directed graph $G' = (V, E')$ by replacing each undirected edge $\{u, v\}$ in G with two directed edges (u, v) and (v, u) in G' while keeping the set of vertices same in both the graphs.

For FindNCS_dec, the set of potential charging station vertices $\widehat{CS}$ can be any non-empty subset of V. To solve the problem instance I of Domset_dec, we choose $\widehat{CS}$ in I' as the set of all vertices, i.e., $\widehat{CS} = V$.

Domset_dec checks for a dominating set where the rest of the vertices are directly connected, i.e., connected by path length of 1. On the other hand, FindNCS_dec checks for a set of charging station vertices CS of a graph G where the rest of the vertices $V \setminus CS$ are connected with some vertex in CS through arbitrary-length paths (bounded by length d). In problem instance I', we set $d = 1$.

Thus, the reduction algorithm converts an instance I of Domset_dec to an instance I' of FindD_dec. Clearly, this conversion happens in polynomial time. With this, we prove the following.

Claim: The graph G has a dominating set of size k if and only if the graph G' has a set of charging stations CS, $CS \subseteq V$, of size k such that all vertices in $V \setminus CS$ are directly connected to some vertex in CS.

The claim can be proved as follows. In I', $\widehat{CS} = V$ and $d = 1$. The only difference between I and I' is that G contains undirected edges, whereas G' contains directed edges. However, for any undirected edge $\{u, v\}$ in G, there exists a directed edge (u, v) in G'. Similarly, for any directed edge (u, v) in G' there exists an undirected edge $\{u, v\}$ in G. Now, as both the problems aim to find the answer to whether there exists a dominating set of size k, the claim holds.

Thus, the polynomial-time reduction from Domset_dec to FindD_dec holds, and as Domset_dec is NP-hard, the NP-hardness of FindD_dec is proved.

■

IV. Finding Minimum Battery Threshold is NP-hard

In this section, we prove Problem 2.2 to be NP-hard. The decision version of FindD is as follows:

*Problem 4.1 (*FindD_dec*):* Consider a directed graph $G = (V, E)$, a set $\widehat{CS} \subseteq V$ of potential charging station locations, and a positive integer k which represents the number of charging station vertices in G. Given another positive integer d, does there exist a set of charging station locations $CS \subseteq \widehat{CS}$, $|CS| = k$, such that the maximum distance of a vertex in CS from any vertex $v \in V$ is at most d? We denote an instance of the problem by $\langle G, \widehat{CS}, k, d\rangle$.

We prove NP-hardness of FindD_dec by showing a polynomial-time reduction from a known NP-hard problem *Vertex Cover* (VC). VC deals with covering all *edges*, whereas FindD deals with covering all *vertices* in a graph.

Definition 4.2 (Vertex Cover Problem (VC) [17]): Given a graph $G = (V, E)$, find a minimum-sized subset $S \subseteq V$ that includes at least one end-vertex of every edge in G.

The decision version of VC is as follows.

*Definition 4.3 (*VC_dec*):* Given a graph $G = (V, E)$ and positive integer k, is there a vertex cover of size k in G? We denote an instance of VC_dec as $\langle G, k\rangle$.

Theorem 4.4: **FindD_dec is NP-hard.**

Proof: We show a polynomial-time reduction from VC_dec to FindD_dec. Consider a problem instance $I : \langle G, k\rangle$ of VC_dec. From the instance I, we create an instance $I' : \langle G', \widehat{CS}, k, d\rangle$ of FindD_dec as follows.

As FindD_dec deals with directed graphs, we transform $G = (V, E)$ into a directed graph $G' = (V', E')$. We replace each undirected edge $\{u, v\}$ between a pair of vertices u, v in G with two oppositely directed edges (u, v) and (v, u) between the same pair of vertices in G'. Furthermore, we replace each edge (u, v) in G' with a pair of edges (u, x) and (x, v) by introducing a new vertex x on the

edge. Conceptually, each edge (u, v) in G is associated with exactly one x-vertex in G'. We also set the weights of these edges as: $wt(u, x) = 1$ and $wt(x, v) = 0$. Let the original vertices in V be called as v-vertices and the newly introduced vertices as the x-vertices. Therefore, V' contains v-vertices and x-vertices. We consider only the v-vertices to be in the set of potential charging stations $\widehat{CS}$. Moreover, as VC_DEC deals with covering of adjacent edges, we set $d = 1$ in I'.

Thus, the reduction algorithm converts an instance I of VC_DEC to an instance I' of FINDD_DEC. Clearly, this conversion happens in polynomial time. We show that the above transformation is indeed a reduction by proving the following claim.

Claim: The graph G has a vertex cover of size k if and only if the graph G' has a set of charging stations CS, $CS \subseteq V$, of size k such that all vertices in $V \setminus CS$ are at most one distance away ($d = 1$) from some vertex in CS.

If: Suppose, in I', we obtain a set of charging station vertices CS, $CS \subseteq \widehat{CS}$, such that $|CS| = k$ and all the vertices in $V \setminus CS$ are at most $d = 1$ distance away from some vertex in CS. This implies that besides v-vertices, all the x-vertices are covered by CS. As each edge in E is associated with exactly one x-vertex in E', as mentioned before, covering all the x-vertices in G' implies covering all the edges in G. Hence, CS is a vertex cover of size k in G.

Only If: Suppose, in I, the set $S \subseteq V$ is a vertex cover with $|S| = k$. Then any edge $e \in E$ is covered by some vertex in S. As per the construction of G', any v-vertex not in S or any x-vertex is at most $d = 1$ distance away from some vertex in S. Hence, in G', S is a set of charging station vertices, with $|S| = k$, covering all v-vertices and x-vertices within $d = 1$ distance away from some vertex in S.

Hence, the polynomial-time reduction from VC_DEC to FINDD_DEC holds, and as VC_DEC is NP-hard, FIND_DEC is also NP-hard. ■

V. APPROXIMATION ALGORITHMS

This section deals with designing approximation algorithms to handle NP-hardness of the two variants of CSPP.

A. Approximation algorithm for finding minimum number of charging stations (ncs)

The algorithm FINDNCS_SC_APPROX (Algorithm V.1) approximately solves the FINDNCS problem with complex motion primitives and complex design of the workspace. This algorithm is based on an approximation algorithm for solving the Set Cover problem [16].

Definition 5.1 (Set Cover Problem [17]): Given a set X of n elements and a collection of subsets $\mathcal{F} = \{S_1, \ldots, S_m\}$ of X, find the minimum-cardinality subset $\mathcal{S} \subseteq \mathcal{F}$ such that the *union* of the elements in $\mathcal{S}$ is X.

Algorithm V.1 finds a near-optimal number of charging stations for a given battery threshold (d). With the given set of motion primitives and workspace design, we create a graph G capturing all possible movements of the robot in the workspace. The graph G along with the set of potential charging stations $\widehat{CS}$ and maximum allowable number of transitions d are inputs to FINDNCS_SC_APPROX. The approximation algorithm SET_COVER_APPROX is a technique for finding a minimal set cover from a collection of subsets

Algorithm V.1: FINDNCS_SC_APPROX: An Approximation algorithm to find minimum number of charging stations.

Input:
G: Transition graph $G = (V, E)$.
$\widehat{CS}$: Potential vertices $\widehat{CS} \subseteq V$ for deploying charging stations.
d: Maximum allowable number of steps to reach a charging station from any obstacle-free location.

Output:
CS: The set with the approximately minimum number of charging stations in G.

Functions :
`DFT`(v, l): A function returning the vertices reachable by *depth-first traversal* starting at vertex v, and up to distance l from v.
`Set_Cover_Approx`$(X, \mathcal{F})$: An approximation algorithm to solve the set cover problem.

```
1 function FINDNCS_SC_APPROX (G, CS^, d)
2 begin
3     H(V, E') ← reverse_edge_directions(G)
4     ∀v_i ∈ CS^: Cover_{v_i} ← DFT(v_i, d)
5     Cover ← {Cover_{v_i} | v_i ∈ CS^}
6     Cover* ← Set_Cover_Approx (V, Cover)
7     CS* ← {v_i | Cover_{v_i} ∈ Cover*}
8     return CS*
9 end
```

of the set. We use this algorithm in our approach for finding the minimal set of charging station locations in a workspace.

The first step of our algorithm is to reverse the edges of G. This is done for ease of computation. The robots move towards a charging station from some location in the workspace following the edges in G. However, for each potential charging station, we want to compute the set of locations from where the charging station is reachable within d steps using depth-first traversal. To achieve this, we create a graph $H = (V, E')$ from G, where E' contains the reversed edge for each edge in E. Depth-first traversal with distance d is used to find the covers for the vertices in $\widehat{CS}$. In line 5, the computed covers are stored in $Cover$. The vertices V and the set of covers $Cover$ are passed to a *known* approximation algorithm SET_COVER_APPROX for computing the minimal set cover of V, with a subset of $Cover$. SET_COVER_APPROX returns the minimal set cover of V and stores it in $Cover^*$. From the generated set cover $Cover^*$, we extract the corresponding vertices for each constituent set in $Cover^*$ and store the vertices in CS^* (line 7). CS^* is the approximately minimum-sized set of charging station vertices that covers all the vertices in V. The size of CS^* is returned as the number of charging stations in H and hence in G.

1) Approximation ratio: Given a transition graph $G = (V, E)$, the approximation factor of FIND_SC_APPROX *is* $\ln(|V| + 1)$.

FIND_SC_APPROX computes the set of covers $Cover$ for all potential charging stations exactly. Then the algorithm calls the approximation algorithm SET_COVER_APPROX and passes the graph parameters (V and $Cover$) to it (line 6). There exists an approximation algorithm for the set cover problem with approximation factor $\ln(n + 1)$ [16], n being the total number of elements in the set. Using this algorithm

Algorithm V.2: FINDD_DOMSET_APPROX: An Approximation algorithm for finding the minimum battery threshold.

Input:
G: Adjacency matrix representation of the graph $G = (V, E)$,
$\widehat{CS}$: Potential vertices in V for deploying charging stations.
ncs: Maximum number of charging stations that can be installed in the workspace.

Output:
CS: Set of charging stations.
d: Minimum battery threshold (or, number of transitions).

```
1  function FINDD_DOMSET_APPROX (G, CS^, ncs)
2  begin
3      i ← 1
4      while i ≤ |V| − 1 do
5          Construct G^i = (V, E^i)
6          Construct (G^i)^2 = (V, (E^i)^2)
7          Compute an IVS M of graph (G^i)^2 s.t. M covers V
             and M ⊆ CS^
8          if |M| ≤ ncs then
9              d ← 2 · i
               /* because 2 · i is the maximum
                  edge-weight in the graph (G^i)^2  */
10             return ⟨M, d⟩
11         end
12         i ← i + 1
13     end
14     if i reaches |V| then
15         Print: ncs charging stations are not sufficient.
16     end
17 end
```

as SET_COVER_APPROX, we can achieve an approximation factor of $\ln(|V| + 1)$ for FINDNCS_SC_APPROX.

2) Time complexity: Given a transition graph $G = (V, E)$, running time of FINDNCS_SC_APPROX *is* $\mathcal{O}(|V|^3)$.

In Algorithm V.1. line 4 finds the covers for each vertex in $\widehat{CS} \subseteq V$. To compute the cover for a single vertex, our algorithm applies *depth-first traversal* that runs in $\mathcal{O}(|V| + |E|)$ time. When $\widehat{CS} = V$, our algorithm computes covers for all $|V|$ vertices. This takes $\mathcal{O}(|V|)^3$ time, because maximum value of $|E|$ is $\mathcal{O}(|V|^2)$. In line 5, computing the minimum set cover using SET_COVER_APPROX algorithm takes $\mathcal{O}(|V| \cdot |Cover| \cdot \min(|V|, |Cover|))$ time [17]. In the worst case, our algorithm considers $\widehat{CS} = V$ and computes covers for all $|V|$ vertices, giving the worst-case time complexity as $\mathcal{O}(|V|^3)$. Hence, the time complexity of FINDNCS_SC_APPROX is $\mathcal{O}(|V|^3)$.

B. Approximation algorithm for finding minimum battery threshold (d)

This algorithm is based on the design of an approximation algorithm for solving the Dominating Set problem [16]. We design our algorithm FINDD_DOMSET_APPROX (algorithm V.2) to approximately solve the FINDD problem with complex motion primitives of robots and complex design of workspaces. Algorithm V.2 finds a near-optimal battery threshold (number of transitions) for a given number of charging stations (ncs).

We create the adjacency matrix G representing the graph $G = (V, E)$. To capture different path lengths in graph G, algorithm V.2 converts an i-length path to a i-weight edge in a modified graph $G^i = (V, E^i)$ using matrix multiplication. As denoted, G^i (derived from G) has maximum edge-weight i. Thus, from an unweighted graph, our algorithm creates a weighted graph with edge weights denoting the path lengths in the original graph. Now, the problem of finding the minimum battery threshold d for a given number of charging stations ncs in G is converted to the following: Minimize d for a given value of ncs, such that the graph G^d has a dominating set $\texttt{domset}(G^d)$ for which $|\texttt{domset}(G^d)| \leq ncs$. However, finding domset for a given graph is NP-hard. To circumvent computational intractability, we use the concept of an approximation algorithm (polynomial time) of domset. This requires us to introduce *Independent Vertex Set (IVS)*.

Definition 5.2 (Independent Vertex Set (IVS) [16]): In a graph $G = (V, E)$, *IVS* is a set S of vertices if, for any two vertices in S, there is no edge connecting the two. That means no two vertices in S are adjacent.

Lemma 5.3 ([16]): Given an adjacency matrix G of a graph, let I be an IVS in G^2. Then $|\texttt{domset}(G)| \geq |I|$, i.e., $|I|$ sets a lower bound on $|\texttt{domset}(G)|$.

In connection with the above, our algorithm finds an IVS covering all the vertices to find the approximate value of d (battery threshold) in G. Algorithm V.2 takes the adjacency matrix of graph $G = (V, E)$, potential charging station vertices $\widehat{CS}$, and ncs as input. The loop between lines 4 and 13 iterates for path length starting from 1 up to $|V|-1$. In the i-th iteration, it constructs G^i using matrix multiplication in line 5, computes $(G^i)^2$ in line 6 and then computes an independent vertex set (IVS) M of $(G^i)^2$ such that M covers V (line 7). Also, $M \subseteq \widehat{CS}$ holds, as M represents the set of charging station vertices in the workspace.

Line 7 adopts a greedy approach to iteratively select the member vertices of M, which are selected in the descending order of coverage size of the vertices. For example, the algorithm selects a vertex $v_1 \in V$ with the maximum number of covered vertices, includes v_1 in M, and flags all the covered vertices as $covered$. For the remaining uncovered vertices, the algorithm selects another vertex v_2 in a similar way and adds it to M. This continues until all the vertices in V are covered.

When the number of transitions i increases, the required number of charging stations ($|M|$) decreases. When $|M|$ reaches within the allowable value of ncs for the smallest value of i, the algorithm assigns $2 \cdot i$ to d in line 9. As i is the maximum edge weight in G^i, by the triangle inequality law, $(G^i)^2$ has a maximum edge weight of $2 \cdot i$. Finally, the algorithm returns $d\ (= 2 \cdot i)$ as the minimum battery threshold with at most ncs number of charging stations in G.

1) Approximation ratio: The approximation factor of FINDD_DOMSET_APPROX *(Algorithm V.2) is* 2.

Let us assume that the algorithm finds the required set of charging stations M^*, $|M^*| \leq ncs$, in the p^{th} iteration. Let the *optimal* battery threshold value be d_{opt} for the graph $(G^p)^2$ with at most ncs charging stations. By Lemma 5.3, $|\texttt{domset}(G^p)| \geq |M^*|$ where M^* is an independent set of $(G^p)^2$. Also, in line 7, M^* computes the charging station vertices that cover all vertices in $(G^p)^2$. Thus, M^* is also a dominating set of $(G^p)^2$. Combining the above statements, the required number of charging stations in G^p is greater than or equal to that of $(G^p)^2$. The battery threshold is inversely

proportional to the number of charging stations. Thus, the battery threshold value in $(G^p)^2$ is greater than or equal to the battery threshold value for G^p. Hence, $d_{opt} \geq p$ as the battery threshold value of G_p is p. As the battery threshold d returned by our algorithm is $2 \cdot p$ (due to triangle inequality as mentioned before), we obtain $d \leq 2 \cdot d_{opt}$.

2) Time Complexity: Given a graph $G = (V, E)$, the running time of FINDD_DOMSET_APPROX *is $\mathcal{O}(|V|^4)$.*

In FINDD_DOMSET_APPROX (Algorithm V.2), line 5 does matrix multiplication which runs in $\mathcal{O}(|V|^{2.8})$ time, due to Strassen's technique [17]. Line 7 adopts a greedy approach to calculate M from the set of $\widehat{CS}$. As per the description of *computation of M*, the greedy algorithm computes IVS M covering all nodes in $(G^i)^2$ in $\mathcal{O}(|V|)^3$ time [18]. Inside the while loop, line 7 dominates all other instructions (including line 5) in terms of time complexity. In the worst case, in our algorithm, M is calculated at most $|V| - 1$ times, which takes $\mathcal{O}(|V|)^4$. Hence, worst case time complexity of Algorithm V.2 is $\mathcal{O}(|V|^4)$.

VI. EVALUATION

A. Experimental Setup

The experiments were carried out in an Intel Core i7-6500U processor with 2.50 GHz clock speed and 16 GB RAM. The algorithms are implemented in C++. The SMT-based algorithms, with which our algorithms are compared, are implemented using Z3 SMT solver [19].

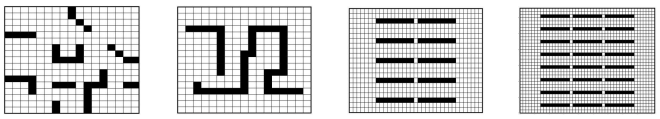
B. Baselines

We compare the performance of FINDNCS_SC_APPROX and FINDD_DOMSET_APPROX with two Satisfiability Modulo Theory (SMT) based algorithms proposed in [15]. The first algorithm SMT_BRUTE_FORCE is based on a *brute-force* approach. Although this approach provides an *optimal* solution, it scales poorly for larger workspaces. The second algorithm SMT_UCORE uses an incremental approach and leverages the *unsatisfiable core* of the constraints. This algorithm provides higher scalability compared to the brute-force approach. However, it does not guarantee an optimal solution. Also, to improve the quality of the solution, SMT_UCORE perturbs the charging station locations, obtained in preceding iterations, to some neighboring locations up to distance $\delta \in \mathbb{Z}^+$. In this work, we use the baseline algorithms SMT_BRUTE_FORCE and SMT_UCORE with $\delta = 0$ (no perturbation) and $\delta = 3$.

C. Experimental Results

We carry out experiments with five workspaces and three robot dynamics. The workspaces are Artificial floor 17×17, Maze 17×17, Warehouse 17×17, Warehouse 35×21, and Warehouse 46×33, as shown in Figure 2. The robot dynamics are of Turtlebot, Dubins vehicle, and Quadcopter, with 9, 16, and 57 motion primitives, respectively.

Our polynomial time approximation algorithms are theoretically proven to produce results that are not worse than the optimal solution beyond certain bounds. Experimental results validate the claims and show that our algorithms can successfully solve many more instances of the NP-hard FINDNCS and FINDD problems, which the SMT-based techniques could not solve.



(a) Art.floor 17×17 (b) Maze 17×17 (c) Warehouse 35×21 (d) Warehouse 46×33

Fig. 2. Workspaces used in our experiments. Warehouse 17×17 is not shown here as it is already shown in Figure 1.

1) Finding ncs: Given the workspace, robot dynamics and battery threshold, FINDNCS_SC_APPROX finds the near-optimal number and locations of charging stations (Table I). SMT_BRUTE_FORCE produces optimal results if it does not time out. Compared to the optimal values of ncs (by SMT_BRUTE_FORCE), our algorithm produces ncs values close to the optimal and within the claimed approximation ratio. However, while the brute force algorithm times out (in 3h) for most of the instances, our algorithm executes in $7\text{s} - 102\text{s}$ (Table I). Table I also shows that for most of the instances, our algorithm FINDNCS_SC_APPROX provides better results than the SMT_UCORE algorithms. Our algorithm provides better quality results compared to the SMT_UCORE with δ=0 for all the instances. SMT_UCORE with δ=3 generally provides results closer to the optimal solutions. Our algorithm provides solutions at par or even better in most of the cases compared to SMT_UCORE with δ=3. In terms of scalability, our algorithm provides a significant speedup over the SMT_UCORE algorithms.

2) Finding d: Table II compares the performance of FINDD_DOMSET_APPROX with the SMT-based algorithms. Given the workspace design, obstacles, and the number of charging stations (ncs), FINDD_DOMSET_APPROX finds the near-optimal value of the battery threshold (d). SMT_BRUTE_FORCE provides an optimal solution, though it times out often. Our algorithm provides solutions close to the optimal and satisfy the approximation ratio derived in this paper. Also, FINDD_DOMSET_APPROX performs at par with SMT_UCORE (δ=3) in terms of quality of the solutions while providing a speedup of up to $\approx 200\times$ over SMT_UCORE with δ=3 (Table II).

3) Performance for a larger workspace: Our algorithms perform well with large workspaces. Table III shows experimental results for a larger Warehouse workspace of size 46×33 and a wide range of values for the fixed parameters. For different d values, we execute FINDNCS_SC_APPROX with Quadcopter motion primitives, and the corresponding ncs values are shown. We execute FINDD_DOMSET_APPROX for different ncs values with Turtlebot motion primitives and show the changes in the obtained values of d. Both algorithms scale well with increasing values of the fixed parameters.

VII. CONCLUSION

We have formulated two versions of the charging station placement problem and converted them to decision problems in the graph paradigm. We have proved NP-hardness for those problems using two well-known NP-hard problems. To handle the NP-hardness of the problems, we have designed

TABLE I

FIND ncs FOR FIXED d: COMPARISON BETWEEN FINDNCS_SC_APPROX AND SMT BASED TECHNIQUES FOR FINDING ncs. RESULTS FOR DIFFERENT WORKSPACES, ROBOT DYNAMICS AND FIXED $d = 6$. "TO" STANDS FOR TIMEOUT (3 HOURS).

Robot type	Algorithm	Art.floor 17×17		Maze 17×17		Warehouse 17×17		Warehouse 35×21	
		T	ncs	T	ncs	T	ncs	T	ncs
Turtlebot	SMT_BRUTE_FORCE	47m 33s	4	57m 36s	5	47m 16s	4	TO	-
	SMT_UCORE (δ=3)	6m 34s	4	18m 34s	6	6m 26s	4	125m 20s	11
	SMT_UCORE (δ=0)	1m 54s	8	1m 29s	8	1m 35s	7	14m 33s	12
	FINDNCS_SC_APPROX	0m 43s	5	0m 36s	6	0m 43s	5	1m 36s	11
Quadcopter	SMT_BRUTE_FORCE	TO	-	TO	-	TO	-	TO	-
	SMT_UCORE (δ=3)	117m 56s	3	97m 3s	6	41m 30s	4	77m 12s	10
	SMT_UCORE (δ=0)	13m 36s	5	12m 33s	7	9m 8s	6	88m 22s	10
	FINDNCS_SC_APPROX	0m 34s	3	0m 22s	5	0m 34s	2	1m 42s	5
Dubins vehicle	SMT_BRUTE_FORCE	TO	-	TO	-	TO	-	TO	-
	SMT_UCORE (δ=3)	TO	-	TO	-	TO	-	TO	-
	SMT_UCORE (δ=0)	71m 51s	10	82m 1s	21	56m 6s	14	TO	-
	FINDNCS_SC_APPROX	0m 7s	5	0m 6s	8	0m 7s	8	0m 14s	14

TABLE II

FIND d FOR FIXED ncs: COMPARISON BETWEEN FINDD_DOMSET_APPROX AND SMT BASED TECHNIQUES FOR FINDING ncs. RESULTS FOR DIFFERENT WORKSPACES, ROBOT DYNAMICS AND FIXED $ncs = 7$. "TO" STANDS FOR TIMEOUT (3 HOURS).

Robot type	Algorithm	Art.floor 17×17		Maze 17×17		Warehouse 17×17		Warehouse 35×21	
		T	d	T	d	T	d	T	d
Turtlebot	SMT_BRUTE_FORCE	34m 20s	4	83m 24s	5	106m 12s	5	TO	-
	SMT_UCORE (δ=3)	6m 58s	5	6m 54s	5	10m 13s	6	TO	-
	SMT_UCORE (δ=0)	1m 1s	6	2m 53s	9	0m 53s	6	78m 40s	13
	FINDD_DOMSET_APPROX	0m 4s	4	0m 4s	6	0m 5s	6	0m 17s	8
Quadcopter	SMT_BRUTE_FORCE	TO	-	TO	-	TO	-	TO	-
	SMT_UCORE (δ=3)	32m 2s	4	47m 36s	5	9m 10s	4	TO	-
	SMT_UCORE (δ=0)	16m 27s	8	9m 36s	7	6m 40s	7	TO	-
	FINDD_DOMSET_APPROX	5m 1s	4	3m 47s	4	5m 17s	6	149m 22s	6
Dubins vehicle	SMT_BRUTE_FORCE	TO	-	TO	-	TO	-	TO	-
	SMT_UCORE (δ=3)	122m 58s	5	118m 51s	5	155m 50s	6	TO	-
	SMT_UCORE (δ=0)	24m 21s	8	41m 30s	10	23m 40s	10	TO	-
	FINDD_DOMSET_APPROX	2m 2s	6	0m 46s	6	0m 37s	8	27m 4s	10

TABLE III

EXPERIMENTAL RESULTS FOR WAREHOUSE 46×33 AND VARYING FIXED PARAMETERS FOR THE APPROXIMATION ALGORITHMS.

FINDNCS_SC_APPROX				FINDD_DOMSET_APPROX			
Robot	fixed d	T	derived ncs	Robot	fixed ncs	T	derived d
Quad	2	0m 29s	66	Turtle	1	12m 22s	28
	3	0m 30s	42		3	11m 29s	26
	4	0m 37s	22		5	8m 36s	20
	5	1m 05s	15		10	5m 5s	12
	6	3m 12s	10		15	3m 17s	8
	7	12m 40s	10		20	2m 14s	6

two polynomial time approximation algorithms, theoretically proved the polynomial run-time of our approximation algorithms, and provided a guarantee on the quality of the solutions within a certain factor of the optimal solution. Experimental results validate the theoretical claims about the quality of solutions and the high scalability of our algorithms.

REFERENCES

[1] B. Kannan, V. Marmol, J. Bourne, and M. B. Dias, "The autonomous recharging problem: Formulation and a market-based solution," in *ICRA*, 2013, pp. 3503–3510.

[2] G. P. Strimel and M. M. Veloso, "Coverage planning with finite resources," in *IROS*, 2014, pp. 2950–2956.

[3] T. Kundu and I. Saha, "Energy-aware temporal logic motion planning for mobile robots," in *ICRA*, 2019, pp. 8599–8605.

[4] I. Shnaps and E. Rimon, "Online coverage of planar environments by a battery powered autonomous mobile robot," *IEEE Transactions on Automation Science and Engineering*, vol. 13, pp. 425–436, 2016.

[5] T. Kundu and I. Saha, "Smt-based optimal deployment of mobile rechargers," in *ICRA*, 2021, pp. 8165–8171.

[6] S. Mishra, S. Rodriguez, M. Morales, and N. M. Amato, "Battery-constrained coverage," in *CASE*, 2016, pp. 695–700.

[7] N. Xu, P. Braß, and I. Vigan, "An improved algorithm in shortest path planning for a tethered robot," Tech. Rep., 2012.

[8] S. Kim, S. Bhattacharya, and V. Kumar, "Path planning for a tethered mobile robot," in *ICRA*, 2014, pp. 1132–1139.

[9] S. McCammon and G. A. Hollinger, "Planning and executing optimal non-entangling paths for tethered underwater vehicles," in *ICRA*, 2017, pp. 3040–3046.

[10] I. Kelly, O. Holland, and C. Melhuish, "Slugbot: A robotic predator in the natural world," in *International Symposium on Artificial Life and Robotics*, 2000, pp. 470–475.

[11] D. Wettergreen, P. Tompkins, C. Urmson, M. Wagner, and W. Whittaker, "Sun-synchronous robotic exploration: Technical description and field experimentation," *IJRR*, vol. 24, no. 1, pp. 3–30, 2005.

[12] J. Wawerla and R. T. Vaughan, "Near-optimal mobile robot recharging with the rate-maximizing forager," in *Advances in Artificial Life*, 2007, pp. 776–785.

[13] T. Kundu and I. Saha, "Mobile recharger path planning and recharge scheduling in a multi-robot environment," in *IROS*, 2021, pp. 3635–3642.

[14] T. Gao, Y. Tian, and S. Bhattacharya, "Refuel scheduling for multi-robot charging-on-demand," in *IROS*, 2021, pp. 5825–5830.

[15] T. Kundu and I. Saha, "Charging station placement for indoor robotic applications," in *ICRA*, 2018, pp. 3029–3036.

[16] V. V. Vazirani, *Approximation algorithms*. Springer, 2001.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[18] G. K. Das, M. De, S. Kolay, S. C. Nandy, and S. S. Kolay, "Approximation algorithms for maximum independent set of a unit disk graph," *Information Processing Letters*, vol. 115, no. 3, pp. 439–446, 2015.

[19] L. M. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS*, 2008, pp. 337–340.